ANALISA KINERJA CLUSTER LINUX SUSE 9.0 MENGGUNAKAN PUSTAKA MPICH V.124 TERHADAP APLIKASI PERENDER GAMBAR POV-RAY V.3.1

Azil Adi Permana, A.Benny Mutiara MQN,Brahmantyo Heruseto

Universitas Gunadarma
Jl. Margonda Raya 100, Depok, 16424
e-mail : amutiara@staff.gunadarma.ac.id,brahm@staff.gunadarma.ac.id

Abstrak

Aplikasi yang sangat membutuhkan waktu seperti software perender citra akan sangat terbantun dengan diterapkannya cluster. Untuk melihat kemampuan system cluster dalam merender citra, maka dibuat percobaan untuk melihat efektifitas POV-Ray sebagai aplikasi perender citra saat berjalan di cluster dengan bantuan pustaka MPICH yang akan mempercepat suatu proses.

Kata Kunci : MPICH,POV-Ray,cluster,linux

1. Pendahuluan

sistem cluster dibuat untuk menciptakan suatu sistem komputer yang memiliki waktu pemrosesan yang sangat cepat, dan sumber daya yang sangat tinggi. Penggunaan komputer pribadi (biasa) atau PC menjawab semua masalah tersebut. Jika dibandingkan antara harga PC (secara keseluruhan) dengan harga superkomputer / mainframe, sangatlah jauh lebih murah. Mungkin sedikit banyak diantara kita tahu, bahwa harga mainframe misalnya Cray, luar biasa mahalnya. Di era 70-an dan 80-an dimana mainframe (IBM AS/400 masuk kategori mainframe) cukup terkenal, hanya perusahaan-perusahaan bermodal besar yang sanggup membelinya.

Karakteristik utama pada sistem cluster adalah bagaimana suatu *task* dapat ditangani secara *bersama -sama*. Kata *bersama -sama* di sini berarti adalah suatu hal yang membedakan mekanisme sistem cluster dengan mekanisme jejaring biasa pada umumnya yang sekedar menjalankan program client server. Seringkali ada kerancuan dengan sistem *Distributed Computing*, tapi sesungguhnya bisa diperjelas lagi bahwa sistem cluster lebih difokuskan pada mekanisme dari Sistem Operasi (OS) dan hardware untuk menyatukan seluruh sumber daya pada setiap node, baik itu CPU, memori, disk, dan sebagainya.

Ada dua tipe sistem cluster yang dominan yaitu :

1. *High Perfomance Computing (HPC).* Secara umum, tipe cluster HPC ditujukan pada bagaimana suatu proses komputasi diakselerasi, dengan demikian *task* bias diselesaikan lebih cepat contoh clustering jenis ini adalah openMosix.
2. *High Availability (HA).* Secara umum, tipe cluster ini ditujukan agar program yang dijalankan di atasnya bisa terus berjalan, sekalipun salah satu node hang atau down. Contoh yang paling mudah adalah *web server* Apache yang diset dengan suatu redirector, sehingga jika salah satu server down, server lain bisa mengambil alih.

2. MPICH

MPICH adalah implementasi lanjut dari MPI standard yang sudah dapat di jalankan dalam berbagai jenis sistem. MPICH digunakan untuk meningkatkan kinerja dari sistem mesin cluster. Message passing adalah paradigma yang digunakan secara luas oleh beberapa kelas dari mesin parallel, terutama distribusi memori. Walaupun ada beberapa variasi, konsep dasar dari proses

Unsupported Personality: PCL
komunikasi antara pesan mudah untuk dimengerti. Selama lebih dari 10 tahun, kemajuan yang cukup baik telah di mulai pada aplikasi yang sangat signifikan. Masing – masing vendor telah mengimplementasikan berbagai jenisnya.

Beberapa sistem telah melakukan sebuah message passing system secara effisien dan portabel dalam pengimplementasiannya. Hal ini merupakan waktu yang sangat berharga untuk memulai pendefinisian dari kedua sintak dan semantik dari inti rutin pustaka yang akan sangat berguna untuk penggunaan user dan efisien dalam pengimplementasian pada sebuah sistem komputer yang sangat luas. Dalam mendesain MPI, telah di implementasikan bagaimana menggunakan fitur - fitur yang atraktif dari sejumlah message passing system, daripada memilih satu dari beberapa dan mengadopsinya sebagai sebuah standard. Pustaka digunakan oleh aplikasi program untuk komunikasi antar proses. Program hanya berisi calls untuk message -passing library yang sangat portable, semenjak program tersebut masuk kedalam berbagai lingkungan pengatur proses penjadwalan kerja.

MPI banyak di pengaruhi oleh hasil kerja IBM T.J. Watson Research Center, Intel's NX/2, Express, nCUBE's Vertex, p4 dan PARMACS. Kontributor penting lainnya dari Zipcode, Chimp, PVM, Chameleon, dan PICL. 2.4.1 Spesifikasi Prosedur Prosedur MPI di spesifikasikan menggunakan bahasa yang independen. Argumen dari pemanggilan prosedur ditandai dengan IN, OUT atau INOUT. Yang berarti:

- ada panggilan, tapi tidak mengupdate argumen ditandai dengan IN,
- ada panggilan dan boleh update, ditandai dengan OUT
- ada panggilan baik itu penggunaan, dan update pada argumen ditandai dengan INOUT.

Semantic Terms

Definisi dari MPI dicoba untuk menghindari, untuk kemungkinan terbesar, penggunaan argumen INOUT. Karena penggunaan tersebut adalah *'error-prone'*, terutama untuk argumen skalar. Fungsi MPI yang biasa digunakan adalah argumen IN untuk beberapa proses dan OUT untuk proses yang lainnya. Argumen INOUT, secara sintaknya, digunakan sebagai semantik dan tidak digunakan pada satu panggilan untuk kedua input dan output. Beberapa kasus muncul, ketika nilai argumen di butuhkan hanya oleh sebuah subset proses. Ketika argumen tidak signifikan pada proses lalu nilai yang arbitrary bisa di lewati sebagai argumen. Walaupun tidak terspesifikasi, argument dari tipe OUT atau type INOUT tidak bisa di atas namakan dengan argument lainnya pada prosedur MPI. Sebuah contoh dari alias argumen pada bahasa C seperti berikut ini:

```
void copyIntBuffer ( int *pin, int *pout, int len)
{ int i;
for (i=0; i<len; ++i) *pout++ = *pin++;
}
```

lalu panggilan pada program tersebut seperti pada code berikut

```
int a[10];
copyIntBuffer( a, a+3, 7);
```

Walaupunbahasa C mengizinkannya tetapi dalam penggunaan prosedur MPI tidak diizinkan walaupun telah dispesifikasikan. Ketika mendiskusikan prosedur MPI beberapa semantic berikut digunakan. Yang kedua pertama biasa digunakan untuk operasi komunikasi.

Nonblocik ing: jika prosedur bisa kembali sebelum operasi selesai, dan sebelum pengguna di izinkan untuk menggunakan ulang sumber daya (seperti buffer) di spesifikasikan pada panggilan.

Blocking: jika kembali dari prosedur mengindikasi pengguna di izinkan untuk menggunakan sumber yang dispesifikasikan pada panggilan.

Local: jika penyelesaian dari prosedur bergantung hanya pada proses lokal. Seperti operasi yang tidak membutuhkan komunikasi dengan anggota proses lainnya.

non-local: jika penyelesaian dari operasi membutuhkan eksekusi dari beberapa prosedur MPI pada proses lain. Seperti operasi yang membutuhkan komunikasi berulang dengan proses user lain.

Collective : jika proses pada grup proses membutuhkan prosedur.

Tipe Data

a. Opaque Objects

MPI mengatur sistem memori yang digunakan untuk membuffer pesan dan untuk menyimpan representasi internal pada beberapa jenis objek MPI seperti grup, komunikator, dan tipe data. Memori ini tidak secara langsung bisa mengakes user dan object yang disimpan secara opaque. Ukuran dan bentuknya tidak terlihat oleh user. Objek opaque di akses melalui handles, dimana ada pada tempat user. Prosedur MPI yang beroperasi pada objek Opaque di lewati melalui argument untuk mengakses objek ini. Sebagai tambahan pada panggilan MPI untuk mengakses objek, handles bisa ikut menangani dan membandingkan. Argumen Array Panggilan MPI membutuhkan argumen dimana pada array dari objek opaque, atau handles array. Handles array adalah array reqular dengan masukkan yang menangani objek pada tipe yang sama dalam lokasi consecutive pada array. Ketika array tersebut digunakan, argumen Len tambahan dibutuhkan untuk mengindikasikan sejumlah masukan valid (selama beberapa argumen tersebut bias di gunakan). Masukan tambahan pada awal array; Le n mengindikasi berapa banyaknya, dan tidak di butuhkan ukuran keseluruhannya. Pendekatan tersebut di ikuti oleh beberapa argumen array lainnya.

b. State

MPI menggunakan beberapa tempat argumen dengan tipe state. Nilai dari tipe data di identifikasikan dengan nama, dan tidak ada operasi di definisikan. Sebagai contoh, rutin MPI_ERRHANDLER_SET mempunyai argumen tipe state yang bernilai MPI_ERRORSFATAL, MPI_ERRORSRETURN, dan sebagainya.

c. Named Constans

MPI menghasilkan beberapa tanda yang memiliki arti khusus pada nilai khusus dari argumen tipe dasar. sebagai contoh, tag yang memiliki nilai integer memiliki argumen komunikasi point-to-point, dengan nilai wild-card khusus, MPI_ANY_TAG. Beberapa argumen akan memiliki kisaran nilai regular, dimana sub kisaran dari kisaran dari nilai pada tipe dasar; nilai spesial akan di luar pada kisaran regular. Kisaran dari nilai regular bisa di queri menggunakan lingkungan fungsi inquiry.

d. Pilihan

Beberapa fungsi MPI menggunakan argumen dengan tipe data pilihan (union). Mekanisme untuk menyediakan hal tersebut akan sangat berbeda dari satu bahasa dengan bahasa lainnya.

e.Alamat

Beberapa prosedur MPI menggunakan argumen alamat yang merepresentasikan alamat absolut pada program. Tipe data seperti integer dari nilai yang dibutuhkan untuk menahan beberapa alamat valid pada lingkungan eksekusi.

Proses

Program MPI terdiri dari proses tak dikenal yang mengeksekusi kodenya sendiri pada tipe MIMD. Kode di eksekusi dengan menggunakan proses yang tidak identik. Proses komunikasi melalui panggilan ke komunikasi MPI primitif. Umumnya, masing - masing proses mengeksekusi pada alamatnya sendiri, walaupun implementasi sharing memory pada MPI sangat mungkin.

MPI tidak menjelaskan model pengeksekusian untuk masing – masing proses. Sebuah proses bisa secara sequensial, atau bisa secara multi-thread, dengan kemungkinan eksekusi threads berulang. Kehati - hatian telah di buat pada MPI "thread-safe", dengan menghindari penggunaan status implisit. Interaksi yang di inginkan dari MPI dengan threads adalah bahwa thread berulang bisa di bolehkan untuk mengeksekusi panggilan MPI, dan panggilan de reentrant; pemblokan MPI call blocks hanya thread berulang, dan membolehkan penjadwalan dari thread lainnya.

MPI tidak menyediakan mekanisme untuk menspesifikasikan alokasi inisial dari proses pada komputasi MPI dan bindingnya pada fisik prosesor. Diharapkan vendor akan menyediakan mekanisme untuk melakukan hal tersebut apakah itu waktu load atau waktu pelaksanaannya. Mekanisme tersebut akan membolehkan spesifikasi dari sejumlah inisial yang dibutuhkan proses, kode yang akan di eksekusi oleh proses inisial, dan alokasi proses pada prosessor.

Penangangan Kesalahan

MPI menyediakan user dengan transimisi pesan yang sangat bagus. Pesan di kirim selalu di terima secara baik, dan user tidak harus mengecek pada kesalahan transmisi, time-out, atau kondisi kesalahan lainnya. Dengan kata lain, MPI tidak menyediakan mekanisme untuk menghadapi kesalahan pada komunikasi sistem. Jlka Implementasi MPI ti bangun pada mekanisme yang tidak reliable, maka sudah menjadi tugas implementator dari subsistem MPI untuk mengisulasikan user dari ke tidak rebilitas, atau untuk secara reflek menutup kesalahan sebagai kegagalan. Sesegera mungkin, kesalahan tersebut akan di refleksikan sebagai kesalahan pada panggilan komunikasi yang relevan. Boleh dibilang, MPI itu sendiri tidak menyediakan mekanisme untuk penanganan
kegagalan prosessor.

Sudah barang tentu, program MPI masih bisa memiliki kesalahan. Kesalahan program bisa terjadi ketika panggilan MPI di panggil dengan argument yang salah. Tipe seperti ini akan ada pada implementasi jenis apapun. Pada kenyataannya, kesalahan sumber daya bisa terjadi ketika program exceeds sejumlah sumber daya sistem yang tersedia. Timbulnya kegagalan ini, bergantung dari sejumlah sumber daya yang ada pada sistem dan mekanisme alokasi sumber daya yang digunakan. Hal ini akan sangat berbeda dari sistem ke sistem. Implementasi yang sangat bagus akan menyediakan berbagai batasan pada sumber daya yang penting sebagai jawaban pada masalah portabilitas.

Sebuah parallel program tidak berjalan pada sistem tertutup. parallel harus memiliki sumber daya lain yang teralokasikan, dan prosesnya harus di jalankan dan di atur, dan prosesnya diasumsikan berkomunikasi satu dengan lainnya. Standardisasi MPI menyangkut juga pada aspek komunikasi, tapi tidak menyangkut pada lingkungan aspek eksekusi lainnya. Satu - satunya cara untuk mengkomposisikan lingkungan yang rumit pada tingkat tinggi di lingkungan parallel adalah dengan memisahkan fungsi dari job scheduler, process manager, message -passing library, dan security.

Job Scheduler

Adalah bahwa sebagian dari sistem yang mengatur sumber daya. Memutuskan prosesor mana yang akan dialokasikan ke suatu pekerjaan parallel ketika berjalan dan ketika akan berjalan. Pada lingkungan lain, hal ini menunjukkan dengan komponen dari sistem antrian batch yang sangat "sophisticated". pada sisi lainnya, direpresentasikan oleh penggunanya sendiri, yang bisa menjalankan pekerjaan kapanpun dan dimanapun dia (pengguna) menyukainya.

Process Manager

Sekali processor telah dialokasikan pada sebuah program, proses user harus di jalankan pada processor tersebut, dan di atur setelah startup. dengan mengatur, hal ini berarti sinya tersebut harus di lalui, yaitu, stdin, stdout, dan stderr harus di tangani pada beberapa cara yang beralasan, dan permintaan untuk pemberhentian suatu tugas bisa di jamin.

Contoh minimalnya adalah rsh, dimana proses penjalanan dan pengaturan stdin,stdout, dan stderr kembali pada proses semula. Contoh rumitnya adalah poe pada IBM SP2 atau prun pada Meiko CS-2, dimana menjalankan proses pada processor yang ditunjuk pada processor tersebut oleh pengatur tugas dan mengaturnya sampai selesai.

Pada beberapa kasus, situasi tergantung pada kombinasi dari fungsi pengatur jadwal kerja dan pengatur proses pada satu jenis software. Sebagai contoh pada pendekatan ini adalah Condor, DQS, dan LoadLeveler. Namun seharusnya, akan lebih nyaman jika memisahkan pada bagiannya masing - masing, dimana masing - masing harus berkomunikasi sesamanya, yang nantinya bisa di modifikasi secara terpisah.

Security

Security adalah hal terpenting dalam suatu lingkungan. System security memastikan bahwa penjadwalan tugas tidak mengalokasikan sumber daya pada pengguna atau program yang tidak seharusnya mendapatkannya. Komponen ini berkomunikasi antara satu dengan yang lainnya dan juga kepada pengguna, tetapi waktu dan path dari komunikasi tersebut beragam dari satu lingkungan ke lingkungan lainnya. Sebagai contoh, job scheduler dan process manager harus berkomunikasi jadi process manager bisa tahu kapan harus menjalankan processnya.

Process manager dan pustaka message-passing saling berkomunikasi untuk mengetahui kapan process akan dijalankan dan bagaimana menghubunginya. Pengguna akan berinteraksi hanya dengan job scheduler (sebagaimana pada kasus LoadLeveler, penjadwalan IBM), secara langsung dengan process manager (poe,prun), atau hanya dengan program aplikasi (p4).

Akhirnya, hal ini akan menjadi berguna untuk program aplikasi untuk secara dinamis meminta lebih banya sumber daya dari job scheduler.

Aplikasi yang membutuhkan komunikasi langsung dengan Runtime

Spesifikasi MPI yang ada di adequasikan untuk hampir seluruh aplikasi. Pada aplikasi tersebut, job scheduler adan process manager, apakah itu sederhana atau rumit, mengalokasikan sumber daya dan menga tur proses user tanpa berinteraksi dengan program aplikasi. Pada aplikasi lainnya, adalah suatu keharusan dimana aplikasi pada level user berkomunikasi dengan job scheduler dan process manager.

Task Farming

Dengan aplikasi "task farm" berarti bahwa program yang mengatur eksekusi satu dengan lainnya, secara sequensial, program. Situasi ini selalu tiba ketika satu ingin berjalan pada program sequensial yang sama pada beberapa waktu dengan berbagai macam input data. Kita panggil masing - masing dari tugas program sequensial. Hal ini bahkan lebih sederhana pada memparalelkan program sequensial yang sudah ada dengan menulis program parallel "harness" yang dipisah, dan prosesnya transien pada masing - masing task. ketika satu task selesai, sebuah proses di mulai kembali untuk mengerjakan selanjutnya. Bahkan ketika sumber daya yang teralokasikan pada tugas fixed, proses tersulit harus berinteraksi dengan process manager. Pada beberapa kasus, hal ini bisa di tulis pada sebuah bahasa scripting sederhana seperti csh atau perl. tetapi kebanyakan pengguna mereferensikan Fortran atau C.

Beberapa nomor dari proses pekerjaan parallel

Program berharap untuk memutuskan di dalam program untuk menyesuaikan sejumlah proses untuk memenuhi ukuran suatu masalah. Selain itu, hal tersebut akan melanjutkan untuk menambah dan mengalikan proses selama perhitungan untuk menyesuaikan fase yang terpisah pada komputasi, beberapa diantaranya mungkin lebih parallel dari yang lainnya. Dari pada melakukan hal ini, program aplikasi akan berinteraksi dengan penjadwalan kerja (sebagaimana yang diimplementasikan) untuk meminta dan memenuhi atau mengembalikan sumber daya komputasi. Hal ini juga akan berinteraksi dengan process manager untuk meminta proses tersebut dimulai, dan dari pada untuk membuat proses yang di ketahui pada message-passing library jadi grup terbesar dari proses bisa berkomunikasi.

Client/Server

Pada situasi ini, kebalikan dari situasi yang sebelumnya, dimana proses datang dan pergi berda sarkan permintaan. Pada model client server, satu set dari proses permanen secara relatif. Pada waktu yang tidak bisa di prediksi, beberapa program memulai pengeksekusian dan harus membuat komunikasi dengan server. Pada kasus ini process manager harus menyediakan sebuah jalan untuk client untuk lokasi server dan komunikasi pada message -passing library dimana hal tersebut harus mendukung komunikasi diantara beberapa proses komunikasi baru.

3. POV-Ray

POV-Ray kependekan dari "Persistence of Vision Raytracer". POV-Ray merupakan sebuah software untuk membuat grafis komputer dengan kualitas yang sangat tinggi. POV-Ray berlisensi freeware, yang berarti setiap orang bisa menggunakannya, dan memperbanyaknya tanpa biaya sedikitpun.

Yang membuat POV-Ray berbeda dari yang lainnya adalah POV-Ray memiliki kekuatan dan kestabilan untuk memuaskan para pengguna yang sangat kompeten dan berpengalaman, meskipun tidak terlalu sulit bagi para pengguna biasa. Tentu saja faktor terpenting adalah kualitas gambar, dan sesungguhnya POV-Ray memiliki hal tersebut.

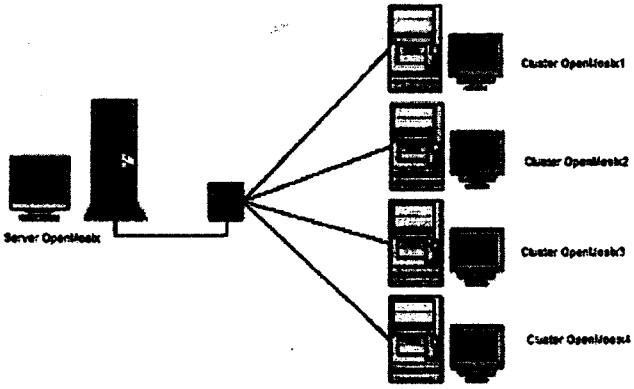
Menurut para pengembang, mereka melihat gambar yang mereka render menggunakan POV-Ray, yang menurut dugaan pertama mereka merupakan fotografi, sangat realistik. Perlu diketahui bahwa foto-realisme merupakan kealilian tingkat tinggi, yang memerlukan banyak latihan untuk menguasainya. Untuk menulis dan memodifikasi scene POV-Ray, pengguna hanya mengedit file teks sesungguhnya yang berisikan perintah - perintah. Pengguna baru mungkin terkejut untuk mempelajarinya, dimana mungkin kedengarannya primitive, hal ini merupakan salah satu fakta yang menunjukkan POV-Ray memiliki kekuatan dan fleksibilitas yang sangat tinggi.

Ada beberapa program untuk merender yang memberikan pengguna fasilitas antar muka point-and-click, tetapi ketika dihadapkan pada keharusan dalam kontrol mutlak suatu scene, hal tersebut sangat sulit jika dibandingkan dengan yang berbasiskan teks, walaupun hal tersebut sangat sulit untuk dipelajari. Hampir setiap orang bisa menggunakan POV-Ray untuk Unix. Jika Anda telah melihat program ray tracing sebelumnya, Anda bisa memiliki kesenangan hanya dengan merender scene contoh dan animasi yang terinsal bersamaan didalam POV-Ray untuk Unix.

Memulai dalam merender POV-Ray beberapa file scene merupakan hal yang sederhana, sesederhana menjalankan "POV-Ray" pada perintah baris dengan diikuti argumen nama filenya. Hal ini akan bekerja baik itu dengan file POV ataupun dengan file INI (selama memiliki hubungan dengan file POV).

4. Pengujian Cluster Linux

Pada pengujian menggunakan sistem Cluster Linux SuSE 9.0 dengan memanfaatkan pustaka MPICH versi 1.2.4. Pengujian dilakukan terhadap aplikasi perender gambar POV-Ray versi 3.1. Dalam pengujiannya menggunakan arsitektur jaringan TCP/IP dalam menyatukan masing – masing node menjadi sebuah sistem cluster. Teknologi jaringan yang dipakai menggunakan topologi star, dipadukan dengan satu buah switch dan konektor RJ45 pada ke lima buah PC.



Gambar1. Arsitektur Jejaring Cluster Linux

Pada gambar arsitektur jejaring komputer Cluster Linux diatas, jejaring Cluster Linux difokuskan pada mekanisme dari Sistem Operasi (SO) dan perangkat keras (hardware) untuk menyatukan sumber daya pada setiap node (CPU, memori, disk, dsb). Dalam teknologi Cluster Linux ada suatu mekanisme tentang bagaimana suatu tugas ditangani secara bersama-sama. Kata bersama-sama disini yang membedakan cluster dengan mekanisme jejaring biasa yang menjalankan program client server atau sistem terdistribusi.

Spesifikasi komputer jejaring Cluster Linux Server dan Client
- Processor AMD Duron 1300 MHz On Board Motherboard ECS K7SOM+
- Memori DDR 256 MB (Server) 128 MB (Client)
- Hard Disk 40 GB 5400 RPM

- LAN Card 10/100 MBPS
- Switch 100 MBPS
- VGA On board 32 MB share memory

Pada jejaring Cluster Linux ini, menggunakan spesifikasi hardware yang sama antara komputer (pc) client dan server. Sedangkan spesifikasi software dibedakan dengan tidak menggunakan fasilitas X-Windows pada komputer client. Hal ini di karenakan, selain tidak adanya manfaat yang akan dihasilkan jika menggunakan fasilitas X-Windows pada komputer client, juga gambar hasil render tidak dihasilkan pada komputer client. Komputer client hanya ditugaskan untuk membantu proses dalam merender, sedangkan hasil dari render di simpan pada komputer server (master). Fasilitas X-Windows pada komputer server (master) akan sangat berguna, terutama dalam melihat langsung gambar dari hasil render POV-Ray.

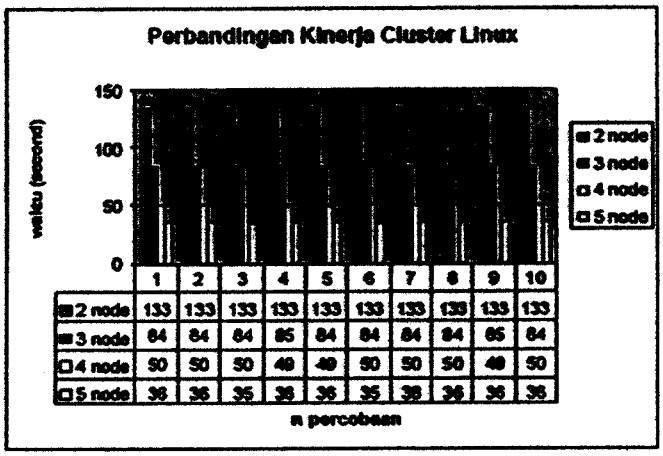
Untuk melihat ke-efektifitasannya, digunakan metode perbandingan antara penggunaan Cluster dengan dua (2) PC, tiga (3) PC, empat (4) PC, dan lima (5) PC. Adapun data hasil pengujian terhadap kedua bentuk jejaring cluster tersebut adalah

Tabel 1. Hasil Pengujian Cluster Linux

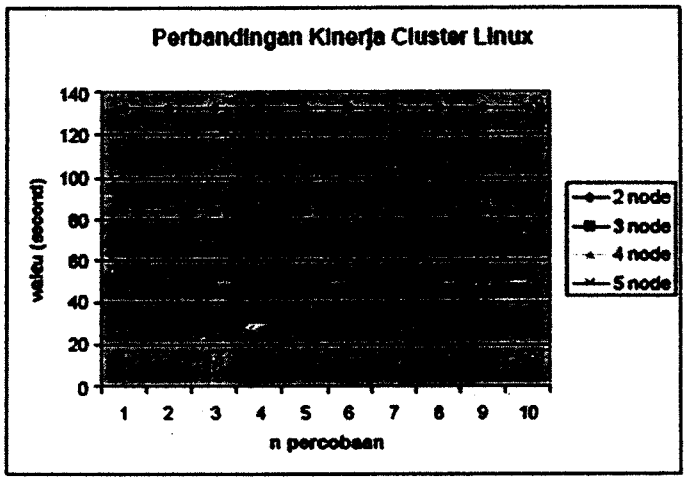
Jumlah Pengulangan Percobaan	Waktu (detik) 2 PC	Waktu (detik) 3 PC	Waktu (detik) 4 PC	Waktu (detik) 5 PC
1	133	84	50	36
2	133	84	50	36
3	133	84	50	35
4	133	85	49	36
5	133	84	49	36
6	133	84	50	35
7	133	84	50	36
8	133	84	50	36
9	133	85	49	36
10	133	84	50	36
Jumlah	*1330*	*842*	*497*	*358*
Rata - rata	*13.3*	*84.2*	*49.7*	*35.8*

Hasil Pengujian Perbandingan Cluster Linux

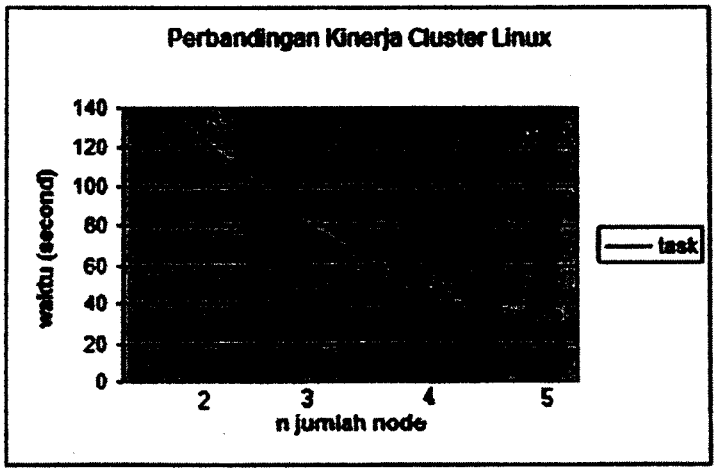
Tabel 2. Perbandingan Antara Jumlah Percobaan Dengan Jumlah Node



	1	2	3	4	5	6	7	8	9	10
2 node	133	133	133	133	133	133	133	133	133	133
3 node	84	84	84	85	84	84	84	84	85	84
4 node	50	50	50	49	49	50	50	50	49	50
5 node	36	36	35	36	36	35	36	36	36	36



Tabel 3. Perbandingan Antara Jumlah Task Dan Jumlah Node



Dari data diatas, terlihat perbedaan waktu dalam memproses gambar menggunakan POV-Ray. Perbedaan waktu tersebut bergantung dari jumlah node (PC) yang digunakan. Semakin banyak node yang dipakai, semakin cepat waktu dalam proses merender gambar POV-Ray. Pustaka MPICH dalam Cluster Linux sangat mempengaruhi kinerja dari Linux Cluster.

Penggunaan tersebut terutama berpengaruh terhadap aplikasi yang membutuhkan sumber daya dan kekuatan yang sangat tinggi. Hal tersebut tentunya digunakan untuk mempercepat dalam memproses suatu tugas (task), dalam hal ini merender gambar 3D berkualitas tinggi.

Salah satu faktor yang dapat mempengaruhi dari kinerja Cluster Linux dalam menggunakan MPICH adalah dengan adanya dukungan untuk pe manggilan *POSIX sched_yield.* Pustaka tersebut menaikkan performa secara signifikan terhadap *ch_shmem*, dan *ch_p4* ketika jumlah total proses lebih banyak (termasuk didalamnya Sistem Operasi dan proses user) daripada jumlah total prosesor. Sedangkan ketika jumlah proses lebih sedikit dari pada jumlah prosesornya, hal tersebut bisa di nonaktifkan dengan menggunakan perintah *–disable-yield.*

5. Kesimpulan

Dari hasil yang diperoleh dalam pengujian Cluster MPICH maka dapat diambil kesimpulan bahwa :

1. Cluster MPICH dapat sangat lebih efisien dalam hal mendapatkan kecepatan rendering gambar 3D oleh POV-Ray jika dibandingkan dengan sistem non Cluster.
2. Dalam melakukan proses komputasi clustering MPICH, dapat di pilih node mana yang di tugaskan untuk proses komputasi, dan node mana yang tidak ditugaskan.
3. Mengingat beratnya aplikasi POV-Ray jika dijalankan pada sistem komputer biasa, maka POV-Ray cocok untuk diterapkan pada sistem Cluster MPICH.
4. Penggunaan node yang sedikit, tidak memberikan begitu banyak pengaruh dalam memperoleh kecepatan. Diperlukan sejumlah node yang cukup banyak untuk mendapatkan kinerja dan kecepatan yang sangat tinggi.
5. Semakin banyak node yang diberikan maka waktu yang dibutuhkan untuk melakukan proses rendering citra lebih singkat.
6. Jumlah node yang digunakan dalam sistem Cluster MPICH dapat disesuaikan dengan kebutuhan seberapa berat aplikasi yang akan di gunakan.

6. Daftar Pustaka

[1]. Michael J. Quinn, **Parallel Computing : Theory and Practice Second Edition**, 1994.
[2]. Mulyadi Santosa, **Clustering di Linux dengan OpenMosix** Email : A_Mulyadi@telkom.net
[3]. Ted G. Lewis & Hesham El-rewini, **Introduction to Parallel Computing** , Prentice - Hall Internasional Editions, 1992.
[5]. Dokumentasi POV-Ray , URL: http://www.povray.org
[6]. Dokumentasi instala si Cluster dengan OpenMosix, URL : http://www.clustering.org
[7]. Dokumentasi kompilasi dan konfigurasi kernel, URL : http://www.kernel.org
[8]. Mailing list kelompok pengguna Cluster Indonesia, URL : clustering-id@yahoogroups.com
[9]. MPI: A Message Passing Interface Standard, Message Passing Interface Forum , 31 Maret, 1994.